IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: Vierich, Ralf and Ferguson, Kevin

Serial No.	: **10/624490**	Group Art Unit :	**2166**
Filed	: **July 23, 2003**	Examiner	: **Ahn, Sangwoo**
For	: **Parameterised Database Drill-through**		
Date	: **February 13, 2007**	Docket No.	:

The Honorable Commissioner of Patents and Trademarks,
WASHINGTON, D.C.
UNITED STATES OF AMERICA 20231

DECLARATION UNDER 37 CFR 1.131

Sir:

The undersigned, being named inventors of the subject Application, declare and state the following:

1. While working for our employer, Cognos Incorporated, the Assignee of the above-identified U.S. Application, we conceived of and completed the invention described and claimed in the subject Application, in Canada, prior to December 17, 2001, the priority date of US Patent Application 10/321,781, published as US Patent Application Publication 2004/0034615 by Neil Thomson, Andre Paiement, Dave Gould, Martin Peticlerc, Brian Donnelly, and Gordon Chow, titled "Universal drill-down system for coordinated presentation of items in different databases" cited in the Office Action mailed October 2, 2006.

2. Attached hereto as *Exhibit A* is a discussion paper by Kevin Ferguson, titled "Parameters & Prompts", created on July 31, 2001 and updated

throughout to January 15, 2002. The invention as claimed was discussed, for example, at page 2, Section 1; page 7, Section 3.2.1; page 8 Section 3.2.2; and page 10, Section 4.

3. Attached hereto as *Exhibit B* is a screen capture of the properties of the discussion paper “Parameters & Prompts”, indicating the Author, Revision Number, Date Created, Date Last Saved for the document.

4. Attached hereto as *Exhibit C* is a design document by Ralf Vierich, titled “Baltic Drill-through Proposal”, created on January 25, 2002 and updated throughout to February 12, 2002. The invention as claimed was discussed, for example, at page 2, 1st paragraph; page 3, 1st paragraph and page 4, 1st paragraph.

5. Attached hereto as *Exhibit D* is a screen capture of the properties of the design document “Baltic Drill-through Proposal”, indicating the Author, Revision Number, Date Created, Date Last Saved for the document.

6. Attached hereto as *Exhibit E* is a document titled “Parameterized drill-through using data expressions within a static drill-through Model” by Ralf Vierich and Kevin Ferguson. The invention as claimed was discussed, for example, at page 1, paragraphs 1 - 3; and page 4, paragraphs 1 - 3.

7. Attached hereto as *Exhibit F* is a screen capture of the properties of the design paper “Parameterized drill-through using data expressions within a static drill-through Model”, indicating the Author, Revision Number, Date Created, Date Last Saved for the document.

8. Both named inventors, Ralf Vierich and Kevin Ferguson, jointly invented the subject matter of all the claims.

As a named inventors, we hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the Application or any patent issued thereon.

Ralf Vierich

K.M. Ferguson

Kevin Ferguson

KMF RV

Date February 13, 2007

Cognos Reporting 1
Discussion Paper

Title: Parameters & Prompts

Document ID:	**kf745**
Version:	**1.1 [4]**
Last Saved:	**2001-08-13 15:42:00**
Document Status:	**Draft**
Author:	**Kevin Ferguson**
Product/Release	

Abstract: (Comments) Proposal for Parameter substitution support in CR-1. Applicability of Prompt specification is briefly discussed.

This document contains information that is deemed proprietary to Cognos. It may not be distributed outside of Cognos without the prior consent of the Vice President of Product Development.

Detailed History of Changes

Ver.	Date	Who	Detailed description of Changes
1.1	2001-08-11	KF	Updated with new thoughts.
1.0	2001-07-31	KF	Created.

Note: New records are placed at the **top** of this list, so that the most recent change is visible on the cover page.

List of Contributors

Initials	Name	Extension
KF	Kevin Ferguson	3212

Cognos Reporting 1

Parameters & Prompts

1. INTRODUCTION

Prompting, drilling [through, down, across...] and [easy] filtering are all symptoms of the same disease: applying filtering criteria to a report. Add the requirement to modify a report specification before it is executed, and you have the short definition of a parameter.

This paper attempts to define parameters that define filter criteria and modify specifications. It will also define prompts and distinguish them from parameters.

1.1 Requirements

Baltic Requirements describe prompts and parameters interchangeably (this paper will distinguish between them below). It specifies that parameters [prompts] must be available for use in report query filters, as well as security-by-value filters on model entities, attributes, tables and columns. In addition, various compnents of a report specification must allow for parameterization so that they can be determined at run-time, including the connection to the metadata model or database, database tables used in SQL queries, output style sheets to be applied, etc. A parameter must accept and apply zero, one or many values as provided.

Baltic Requirements also specify that prompting the user must be optional (all values will be selected), default value(s) may be provided from a calculation or query, and any type-in values must be formatted and/or verified. Prompts may depend upon other parameters [prompts], thus defining cascading prompts.

1.2 Proposal

In order to satisfy the above Requirements, prompts will be implemented to solicit values from the user and parameters will be implemented to manage the filter criteria or specification modifications that arise from the entered/selected values. Parameter support will be designed such that the prompt definitions are not required or may be overridden via explicit provision of desired values.

The distinction between parameters and prompts is subtle, but important. It may help to consider a prompt as a value *provider* and a parameter as a value *consumer*. Prompts are designed to provide values for selection or accept entered values -- either the user or another application can be "prompted". OTOH, a parameter definition is used to convert the entered or selected values into filters and specifications that may be applied to a report.

It is always possible to derive a prompt definition from a parameter specification -- in fact; it may be advantageous to combine the two specifications. However, since a parameter specification requires a metadata reference such as a query item or entity attribute, it is not always possible to derive a parameter from a prompt specification (especially a type-in prompt).

2. Functional Specification

2.1 Parameters

A CR1 parameter is a combination of prompt definition and [implicit] filter specification. A parameter specification is composed (in part) of a reference to a metadata component, usually a query item, an entity attribute, or an expression. References to one or more [filter] expressions and/or queries may also be specified so that the report author may direct the application of the values that are provided at run-time. Prompt specifications are derived from the referenced metadata component(s).

Usually, filter expressions are generated from parameter definitions and any values that are provided at run-time. Substitution of Parameter Values in an expression is also possible, although the syntax for such a specification has yet to be finalized. [Impromptu uses "?Px1?" syntax for a prompt named 'Px1']

Unlike prompts in Impromptu, CR1 parameters may be optional – user is not required to enter/select run-time values; however, the report author can enforce the use of any parameter. When optional, the parameter may be ignored, in which case filter expressions are neither generated nor applied to the report queries. If a query filter refers to an optional parameter that is ignored by the user, the filter will be removed from the query.

2.1.1 Examples

Syntax is arbitrary. Identifiers and references displayed in purple.

- Two simple parameter specifications for same CustID query item. The metadata component for each parameter is implied by the query item reference within the expression.

```
<BIQuery id="Q1" > ... <DataItem name="CustID">[Customers.Cust_CD]</DataItem> ... </BIQuery>
<Parameter name="Customers"><Expression>[Q1].[CustID]</Expression></Parameter>

<BIQuery id="Q2" >
    ... <DataItem name="CustID">[Customers.Cust_CD]</DataItem> ...
    ... <Parameter name="Customers"><Expression>[CustID]</Expression></Parameter> ....
</BIQuery>
```

- Mandatory parameter applied to multiple queries: in this case, SQ1, SQ2 and main query. Since this parameter is mandatory (use="required"), the default value will be applied if user ignores prompt.

```
<Parameter name="Time Period" use="required">
    <Expression>[OrderHeader].[OrderDate]</Expression> <!-- metadata component -->
    <QueryReference idQuery="SQ1"/>  <!-- apply default filter to SQ1 -->
    <QueryReference idQuery="SQ2"/>  <!-- apply default filter to SQ2 -->
    <ParmFilter idQuery="main"> <!-- apply this filter to main query -->
        <Condition>([Time Period].[Start] < [QI2] AND [QI2] <= [Time Period].[End])</Condition>
    </ParmCondition>
    <PromptDefault><Range><Start>2001-03-01</Start><End>2001-05-31</End></Range></PromptDefault>
</Parameter>
```

- Optional parameter related to a query filter: if parameter not used, filter is removed from query.

```
<BIQuery id="Qry5">
    ... <Filter name="product">[Products].[Prod_Cd] IN ( [Parm1] )</Condition> ...
</BIQuery>
<Parameter name="Parm1"> ...
    ... <Filter ref="[Qry5].[product]"/> ...
</Parameter>
```

2.2 Prompts

A CR1 Prompt is used to request values from a user. It does not apply the entered/selected values to the report: the parameter does that. Technically, a CR1 prompt is a self-contained report with user input or selection capability. When presented to a user, the HTML control that is generated will depend upon the type of prompt (type-in, pick-list…) and any constraints placed upon it (data type, single/multiple value, ranges…).

Prompting a user is really a reporting exercise with callbacks encoded as URLs. Several prompts will be presented on a single HTML page, with cascading prompts appearing on the same page.

Prompt specifications in CR1 are familiar to any Impromptu user: there are type-in and pick-list prompts. New to CR1 are multiple value type-in prompts, enumerated pick-lists, value ranges, default values and lookup validations. Query definitions are often associated with a pick-list query in order to specify the values that will be presented to a user.

Prompt definitions are associated with parameters in order to override default behaviour for requesting parameter values from a user. The parameter definition can be used to derive a [default] prompt.

2.2.1 Examples

Syntax is arbitrary.

- Type-in prompt

```
<Prompt name="PdEnds">
     <Message>Enter end of period:</Message>
     <TypeInPrompt>
               <PromptDefault><PValue>today()</PValue></PromptDefault>
     </TypeInPrompt>
</Prompt>
```

- Validated Type-in prompt with multi-lingual messages

```
<Prompt name="ProductCode">
     <Message>Enter a product code:</Message>
     <Message lang="pig-latin">Terenay a roductpay odecay:</Message>
     <TypeInPrompt>
          <Lookup ref="[Products].[Prod_Cd]"/>
     </TypeInPrompt>
</Prompt>
```

- Pick-list prompt: [validated] type-in allowed

```
<QueryReference id="PL" src="Product PL.XML"/>
<Prompt name="ProductPicker">
     <Message>Select products:</Message>
     <Message lang="pig-latin">Electsay roductspay:</Message>
     <PickListPrompt type-in="true">
          <PromptDataSource>
               <UseItem ref="[PL].[Prod_Cd]"/>
               <DisplayItem ref="[PL].[Prod_Name]"/>
               <DisplayItem ref="[PL].[Prod_Price]"/>
          </PromptDataSource>
          <Lookup ref="[PL].[Prod_Cd]"/>
     </PickListPrompt>
</Prompt>
```

- Enumerated Prompt

```
<Prompt name="Model Names">
    <Message>Select a model:</Message>
    <PickListPrompt>
        <PromptChoiceList>
            <Choice><Use>GOSales</Use><Display>Great Outdoors</Display></Choice>
            <Choice><Use>GOSales2k</Use><Display>Great Outdoors 2000</Display></Choice>
        </PromptChoiceList>
    </PickListPrompt>
    <PromptDefault><PValue>GOSales2k</PValue></PromptDefault>
</Prompt>
```

2.3 Parameter Values

Whether entered, selected or generated, parameter values that are provided to the CR1 report are substituted into the specification at author-designated locations, or translated into filter expressions and applied to the appropriate queries. In Impromptu, this translation was defined explicitly as prompt references within filter expression by the report author (also available in CR1), which limited the flexibility available to the end-user who is responding to the prompts. In CR1, filters may be generated at run-time, which will allow the user to ignore prompts, enter or select ranges of values and explicitly omit undesirable values or ranges.

Any filter expressions that have been authored for the report will not be affected by the addition of filter values, nor will any applicable user filters. If any contradictions arise between the authored/user filters and those generated from parameter values, the query will return no data (i.e. zero rows). The report author may override this behaviour by associating any conflicting filters with a parameter that also defines default values.

2.3.1 Examples

Syntax is arbitrary.

- Type-in value for a prompt

```
<ParmValueList prompt="PdEnds">
    <PValue>2001-03-01</PValue>
</ParmValueList>
```

- Pick-list values for a parameter: in this case, two date ranges

```
<ParmValueList parameter="Time Period">
    <Range><Start>2001-03-01</Start><End>2001-05-31</End></Range>
    <Range><Start>2000-03-01</Start><End>2000-05-31</End></Range>
</ParmValueList>
```

- Inclusive and exclusive values for a parameter: in this case, product codes 101, 102, 200 up to (but not including) 300, but not 232.

```
<ParmValueList parameter="Parm1">
    <PValue>101</PValue>
    <PValue>102</PValue>
    <Range><Start>200</Start><End use="exclusive">300</End></Range>
    <Pvalue use="exclusive">232</PValue>
</ParmValueList>
```

2.3.2 Localization

Values presented to users for selection must be formatted according to their regional specifications. When the values are selected or entered, they must be decoded (and possibly reformatted) before application to query filters. Specification of the format that is used must accompany the parameter value list, either directly or indirectly via the parameter definition. The format may be implied through a locale specification or defined explicitly, as outlined below:

- Format implied by locale

```
<ParmValueList prompt="PdEnds" locale="en-US">
     <PValue>03-01-2001</PValue>
</ParmValueList>
```

- Format specified explicitly

```
<ParmValueList prompt="PdEnds"><Format>MM-DD-YYYY</Format>
     <PValue>03-01-2001</PValue>
</ParmValueList>
```

- Format inferred from prompt or parameter

```
<Prompt name="PdFmt">
     <Message>Enter start of period:</Message>
     <TypeInPrompt>
          <Format>MM-DD-YYYY</Format>
          <PromptDefault>today()</PromptDefault>
     </TypeInPrompt>
     <DbType type="date"/>
</Prompt>

<ParmValueList prompt=" PdFmt">
     <PValue>03-01-2001</PValue>
</ParmValueList>
```

3. Applying Parameter Values

3.1 Value Substitution

Parameter Values that are applied to a request specification may be substituted directly into the report specification through report expressions. Like any other element in an XML specification, parameters are named objects that contain properties. Such properties include the name, message, value(s), start/end value in a range, etc. Any report expression may refer to these properties wherever appropriate. Therefore, parameter values may be substituted into a specification wherever a report expression is allowed.

3.1.1 Examples

Syntax is arbitrary.

- Prompt for name of model

```
<ModelConnection id="GO"><Name>[Model Names].[value]</Name></ModelConnection>
```

- Display prompt values in a Report Title

```
<PageHeader>
    <TextItem>
        <ReportExpression>"Model: " + [Model Names]</ReportExpression> <!-- .[value] is assumed -->
    </TextItem>
</PageHeader>
```

3.2 Query Filters

Given one or more lists of parameter values as well as the parameter specifications to which they refer, the CR1 report server must generate filter specifications and apply them to one or more queries in the report. The proposed algorithm is divided into two phases: generating filters and applying them.

3.2.1 Generating Filters

Generating an E-R filter is relatively straightforward:

1. segregate the individual values from the ranges into separate sets;
2. within each set, further segregate the inclusive values/ranges from the exclusive ones;
3. for the set of inclusive values, generate a filter IN clause using the metadata reference in the parameter and the inclusive values
   - e.g. [metadata] IN (value1, value2, value3…);
4. for each inclusive range, generate a filter clause between the start and end values
   - e.g. startValue <= [metadata] AND [metadata] <= endValue;
5. conjoin the filter clauses generated in steps 3 and 4 with OR operators;
6. for the set of exclusive values, generate a filter NOT IN clause using the metadata reference in the parameter and the set of exclusive values
   - e.g. [metadata] NOT IN (valueX, valueY, valueZ…);
7. for each exclusive range, generate a filter clause exclusion the start and end values
   - e.g. startValue > [metadata] OR [metadata] > endValue;
8. conjoin the filter clauses generated in steps 5, 6 and 7 with AND operators.

Generating an OLAP filter follows a similar algorithm; however, the final syntax differs somewhat from the E-R filter.

3.2.2 Applying Filters

After Generating Filters (see above), parameter filters must be applied to the queries of the report. A given parameter may not be applicable to every query and sub-query in the report; therefore, applying the parameter's filters blindly to every query would be inappropriate.

In the entity-relationship environment of CR1, a query is based upon one or more attributes and (by inference) entities. Parameters must also refer to metadata components (see above), thus they are also based [indirectly] upon one or more entities. Therefore (by default), CR1 report server will apply a generated filter to any query that is based upon the same entities as the parameter that generated the filter.

When querying directly against a database, the CR1 query is based upon columns and (by inference) database tables. In such queries, the Parameters will be based upon the database columns and/or any query items that refer to them. As such, parameter filters will be applied against queries that are based upon the same database tables as the filter.

The CR1 report author must be able to override this default behaviour. A parameter may contain references to the queries to which its generated filters should be applied (see `Time Period` example above). In addition, actual filters may be identified (see `Parm1` example above).

3.2.3 Examples

E-R filters only. Syntax is arbitrary. All value lists refer to the `Customers` parameter (see above).

- Inclusive individual values: any of 1001, 1010, 1034 or 1010.

  ```
  <ParmValueList parameter="Customers">
       <PValue>1001</PValue>
       <PValue>1010</PValue>
       <PValue>1034</PValue>
       <PValue>1010</PValue> <!- NB: duplicate value ->
  </ParmValueList>
  ```

  Generates the following filter:

  ```
  <condition>[Q1].[CustID] IN (1001,1010,1034)</condition> <!- NB: duplicate value eliminated ->
  ```

- Exclusive individual values: NOT 2021 nor 1098

  ```
  <ParmValueList parameter="Customers">
       <PValue use="exclusive">2021</PValue>
       <PValue use="exclusive">1098</PValue>
  </ParmValueList>
  ```

  Generates the following filter:

  ```
  <condition>[Q1].[CustID] NOT IN (2021,1098)</condition> <!- NB: specification order preserved ->
  ```

- Inclusive range

  ```
  <ParmValueList parameter="Customers">
       <Range><Start>1001</Start><End>1009</End></Range>
  </ParmValueList>
  ```

  Generates the following filter:

  ```
  <condition>(1001 <= [Q1].[CustID] AND [Q1].[CustID] <= 1009) </condition>
  ```

- Multiple inclusive ranges

  ```
  <ParmValueList parameter="Customers">
       <Range><Start>1001</Start><End>1009</End></Range>
       <Range><Start>2031</Start><End use="exclusive">2047</End> <!- NB: exclusive end value -> </Range>
  </ParmValueList>
  ```

Generates the following filter, conjoined by OR operator:

```
<condition>(1001 <= [Q1].[CustID] AND [Q1].[CustID] <= 1009) OR
 (2031 <= [Q1].[CustID] AND [Q1].[CustID] < 2047) <!- NB: exclusive end value (2047) generates 'less than'
instead of 'less than or equal' operator -> </condition>
```

- Exclusive ranges

```
<ParmValueList parameter="Customers">
     <Range use="exclusive"><Start>2052</Start><End>2066</End></Range>
</ParmValueList>
```

Generates the following filter:

```
<condition>(2052 > [Q1].[CustID] OR [Q1].[CustID] > 2066) </condition>
```

- Multiple exclusive ranges

```
<ParmValueList parameter="Customers">
     <Range use="exclusive"><Start>2052</Start><End>2066</End></Range>
     <Range use="exclusive"><Start>2047</Start><End>2061</End> <!- NB: includes other range -> </Range>
</ParmValueList>
```

Generates the following filter, conjoined by AND operator:

```
<condition>(2052 > [Q1].[CustID] OR [Q1].[CustID] > 2066) AND
 (2047 > [Q1].[CustID] OR [Q1].[CustID] > 2061)</condition>
```

- Above examples combined and intermingled:

```
<ParmValueList parameter="Customers">
     <Range use="exclusive"><Start>2047</Start><End>2061</End></Range> <!- NB: exclusive range ->
     <PValue>1001</PValue> <!- NB: inclusive individual value ->
     <Range use="exclusive"><Start>2052</Start><End>2066</End></Range> <!- NB: exclusive range ->
     <PValue>1010</PValue> <!- NB: inclusive individual value ->
     <Range><Start>2031</Start><End use="exclusive">2047</End></Range> <!- NB: inclusive range ->
     <PValue use="exclusive">2021</PValue> <!- NB: exclusive individual value ->
     <PValue>1034</PValue> <!- NB: inclusive individual value ->
     <Range><Start>1001</Start><End>1009</End></Range> <!- NB: inclusive range ->
     <PValue use="exclusive">1098</PValue> <!- NB: exclusive individual value ->
     <PValue>1010</PValue> <!- NB: inclusive individual value [duplicate] ->
</ParmValueList>
```

Generates the following filter:

```
<condition>(2047 > [Q1].[CustID] OR [Q1].[CustID] > 2061) <!- NB: first exclusive range -> AND
([Q1].[CustID] IN (1001,1010,1034) <!- NB: ALL unique inclusive individual values -> OR
 (2031 <= [Q1].[CustID] AND [Q1].[CustID] < 2047) <!- NB: first inclusive range -> OR
 (1001 <= [Q1].[CustID] AND [Q1].[CustID] <= 1009) <!- NB: second inclusive range ->) AND
(2052 > [Q1].[CustID] OR [Q1].[CustID] > 2066) <!- NB: second exclusive range -> AND
[Q1].[CustID] NOT IN (2021,1098) <!- NB: ALL exclusive individual values -> </condition>
```

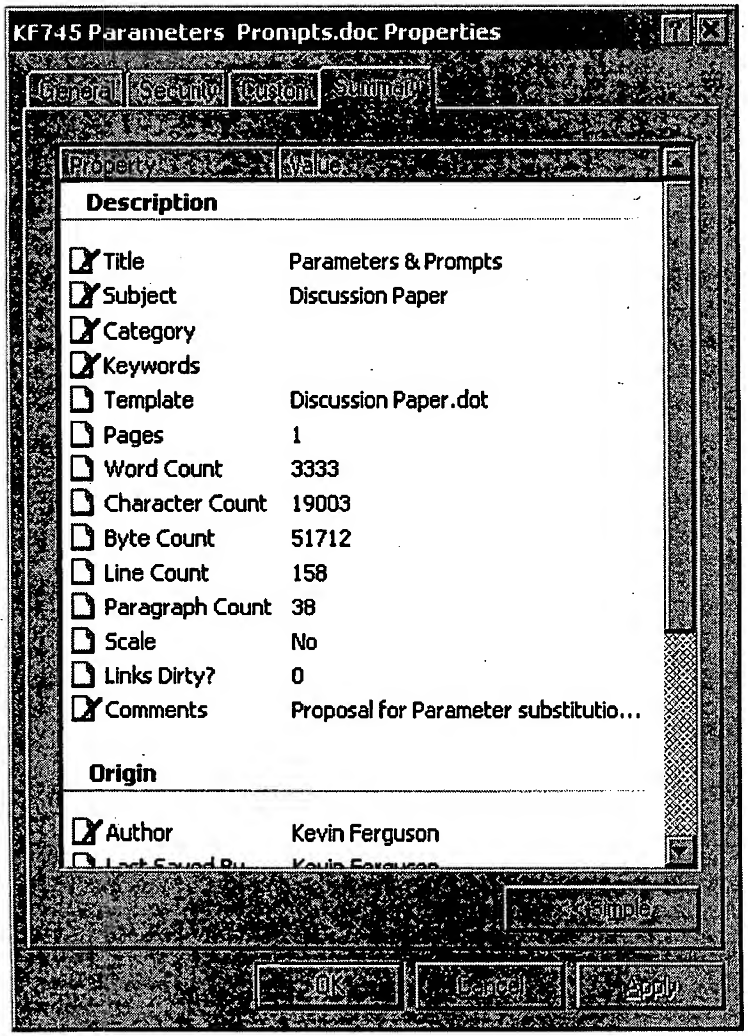
Since individual filters in a collection are conjoined by AND operators, the following is equivalent:

```
<FilterList>
   <condition>(2047 > [Q1].[CustID] OR [Q1].[CustID] > 2061) <!- NB: first exclusive range -> </condition>
   <condition>[Q1].[CustID] IN (1001,1010,1034) <!- NB: ALL unique inclusive individual values -> OR
    (2031 <= [Q1].[CustID] AND [Q1].[CustID] < 2047) <!- NB: first inclusive range -> OR
    (1001 <= [Q1].[CustID] AND [Q1].[CustID] <= 1009) <!- NB: second inclusive range -> </condition>
   <condition>(2052 > [Q1].[CustID] OR [Q1].[CustID] > 2066) <!- NB: second exclusive range -> </condition>
   <condition>[Q1].[CustID] NOT IN (2021,1098) <!- NB: ALL exclusive individual values -> </condition>
</FilterList>
```

All of the above Filters will be applied to Q1 query only, because the Customers parameter is based upon [Q1].[CustID] query item.

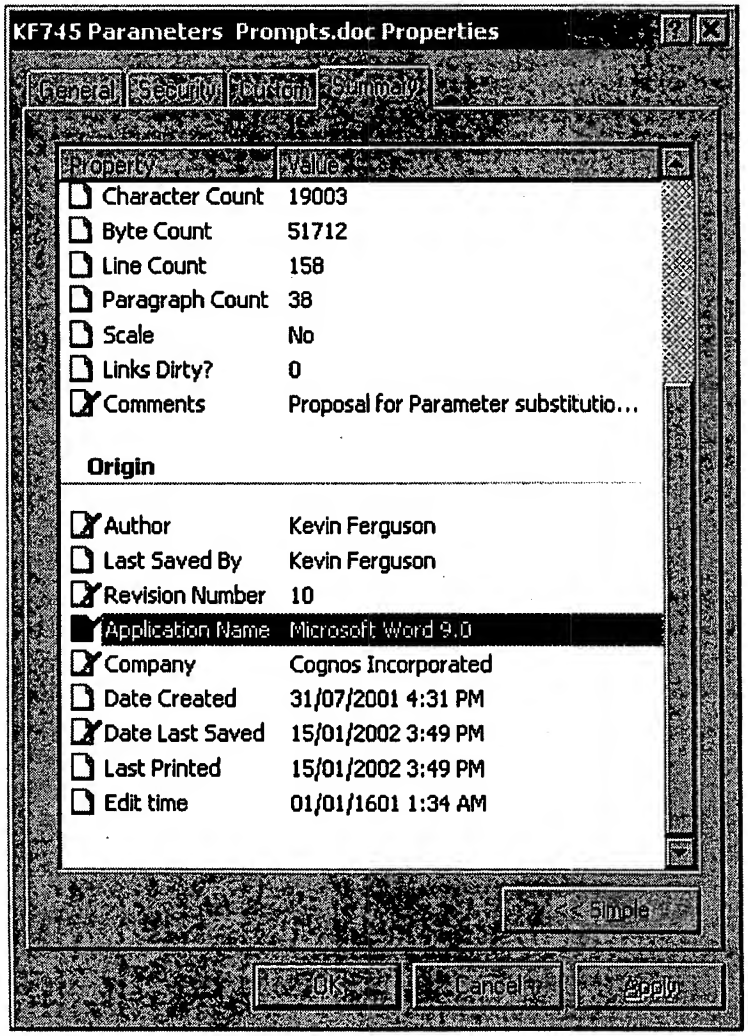
4. SUMMARY

The implementation of parameters, their associated [filter] values, and the prompts that provide the author / report consumer with choices and/or type-in validation have application throughout CR1. First & foremost, prompting a user is enhanced by the additional functionality [multiple values, ranges, exclusion] that a parameter provides. Applying filter criteria to a report is also enhanced, whether that application originates from a drill operation, [easy] filter or a third-party application driving a pre-authored report. What remains to be determined is the metadata content that would facilitate the matching of parameter specifications across application boundaries, such as drill operations between CR1 and PowerPlay.



KF745 Parameters Prompts.doc Properties

Property	Value
Description	
☑ Title	Parameters & Prompts
☑ Subject	Discussion Paper
☑ Category	
☑ Keywords	
□ Template	Discussion Paper.dot
□ Pages	1
□ Word Count	3333
□ Character Count	19003
□ Byte Count	51712
□ Line Count	158
□ Paragraph Count	38
□ Scale	No
□ Links Dirty?	0
☑ Comments	Proposal for Parameter substitutio…
Origin	
☑ Author	Kevin Ferguson
□ Last Saved By	Kevin Ferguson



KF745 Parameters Prompts.doc Properties

Property	Value
□ Character Count	19003
□ Byte Count	51712
□ Line Count	158
□ Paragraph Count	38
□ Scale	No
□ Links Dirty?	0
☑ Comments	Proposal for Parameter substitutio…
Origin	
☑ Author	Kevin Ferguson
□ Last Saved By	Kevin Ferguson
☑ Revision Number	10
☑ Application Name	Microsoft Word 9.0
☑ Company	Cognos Incorporated
□ Date Created	31/07/2001 4:31 PM
☑ Date Last Saved	15/01/2002 3:49 PM
□ Last Printed	15/01/2002 3:49 PM
□ Edit time	01/01/1601 1:34 AM

Exhibit B

Baltic Drill-through Proposal

(Basis of which will form a function specification)
Ralf Vierich in collaboration with Kevin Ferguson (28-Jan-02)

Terminology

CR1

- **Report column** is related to a single query item. A query item is either a reference to a single BMT query element or an expression of one or more query elements

BMT

- **Query element** relates to a database column or an expression that may contain zero or more columns. Query elements are children of **Query Objects**.

Introduction

This document is to propose a high level design for drill-through in the Baltic environment.

Drill-through requirements cover…

- From PPWeb 7.0 to CR1 saved reports both Query Studio and Report Studio
- CR1 report to report
- CR1 report to external URL

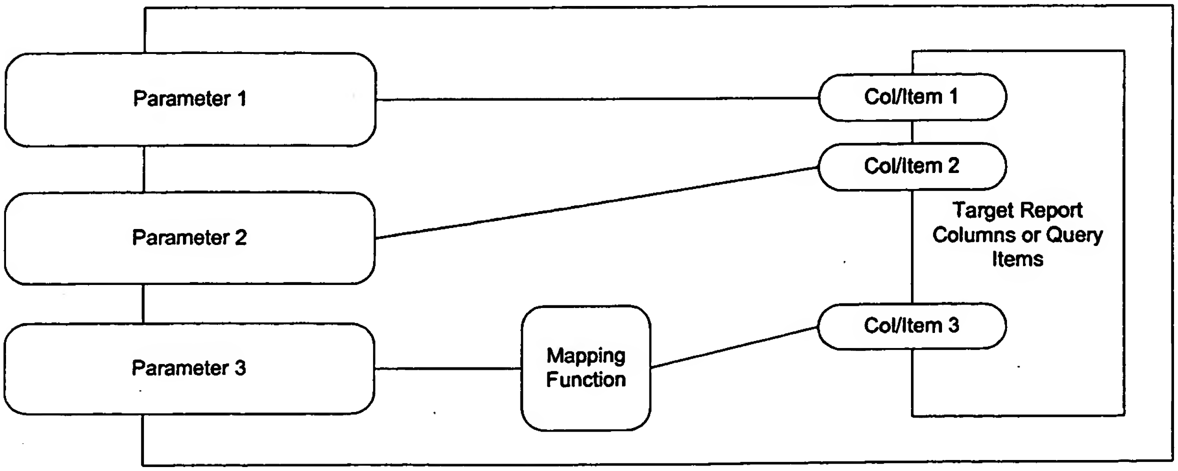
Not required are….

- PowerPlay client (windows) to CR1
- CR1 to PPWeb or any other Cognos product (although covered by CR1 report to external URL)

Drill-through Target

Drill-though from the Source to the Target is accomplished with a "parameter mapping" concept. A drill target conceptually is a collection of parameters that map to the actual target report columns or query items.

A parameter is identified by both it's name and datatype. A parameter may map to more than one Query Element or Report Column and vise-versa.



How a Drill-thru URL uses/relates to Parameters

A drill-thru request into CR1 will be done via two separate URL's (for V1 anyway). One URL is for Report Studio and the other is for Query Studio.
Both URLs may specify a drill-predicate and a target report or query subject.
The drill predicate consists of one or more parameter name and value pairs. The parameters in the drill-predicate are matched up with one defined in CR1 and based on the report or query subject specified in the URL the columns and these are applied as the basis of a filter predicate in the target report.
Here are the two proposed URL's. More detail will be spec'd out later

```
//<servername>/baltic/cgi-
bin/cognos.cgi?b_action=report.run&requestcontent=<reportname>&drill=<drillpredicate>[&<
otherparameters>...]
//<servername>/baltic/cgi-
bin/cognos.cgi?b_action=query.run&requestcontent=<reportname>&drill=<drillpredicate>[&<
otherparameters>...]
```

Parameters

A parameter is a named mapping to column or query element. Attributes of a parameter are...

- Name
- Data type
- Mapping function (optional)
- Required (is the parameter required)
- Default value (optional)
- More **TBD**

More than one parameter can map to the same report column.

Parameter Mapping

It will be possible to perform some kind of conversion from the value specified in the drill predicate before being mapped to the column (ultimately a report predicate). This is done via a mapping function. In the simplest case and V1 deliverable it will be an expression understood by the query engine. So for example a drill predicate may have Country='Germany' but the parameter maps to a report column that required an ISO country code. In this case the mapping function would convert 'Germany to 'DE" (possibly via a 'decode()' function). Another potentially common case would be to convert date-time values to date. A mapping function is optional
In a future release the mapping function could be a JavaScript function or plug-in function (whatever).

Explicitly Defined Parameters

As part of creating a report the report author can define a set of parameters that are to be used when drilling into the report. A UI would need to be developed that allows the user to create the parameters, optionally define mapping functions and assign them to a report column. Explicitly defined parameters are **optional.**

Implicitly Defined Parameters

The report columns are themselves implicit parameters and therefore every report is a potential drill-through target. If the drill-through URL has a drill-predicate whose parameters and datatypes in the drill-predicate match the column names and datatypes of the target report columns then drill-through is possible. Some rules need to be defined as to which columns are eligible to be parameters such as; parameters may not map to blobs or aggregates.

A report column references a CR1 query item that may reference one BMT query element or an expression with zero or more query elements.

In the case where a report column references a single query element then the implicit parameter would be the query element. So, as with the report column case, if the drill predicate parameters name and datatypes match the query element's name and datatype then drill-through can occur. The same rules as eligible parameters would apply to query elements.

The rules that govern what columns are parameter candidates are **TBD**

Model Defined Parameters

So far we have talked about drill-thru parameters as being defined either explicitly or implicitly by the report. What is also required is a more general definition that can be done within the model (meta-data) and reused my all reports. To address this issue drill-through parameter support will be added to the CR1 model.

Parameters can be defined in the model that map directly or indirectly through a mapping function to query elements. This allows the model designer to associate a set of know drill-through parameters to query elements.

For example; Assume there are no explicit parameters defined in the report. A drill through URL predicate has a number of parameter. Since there are no explicit parameters we will look at the query elements that make up each report column. If the query element has one or more parameters defined for it and one matches the one in the drill predicate then the parameter is consider the implicit parameter for the report column.

Parameter Search Order

When a drill-through request is made the drill predicate parameters must be matched up with the report columns. The search order of matching drill-predicate parameters with report columns will be done in the following order...

1. Explicit parameters defined in the report definition
2. Model defined parameters
3. Implicit parameters derived from the report column names

If one or more parameters for the report are marked as required and are not matched up with parameters in the drill-predicate then a warning action is triggered (maybe even is logged in a log file). The same rules as IWR can be used to report warnings.

URL for Drill-through

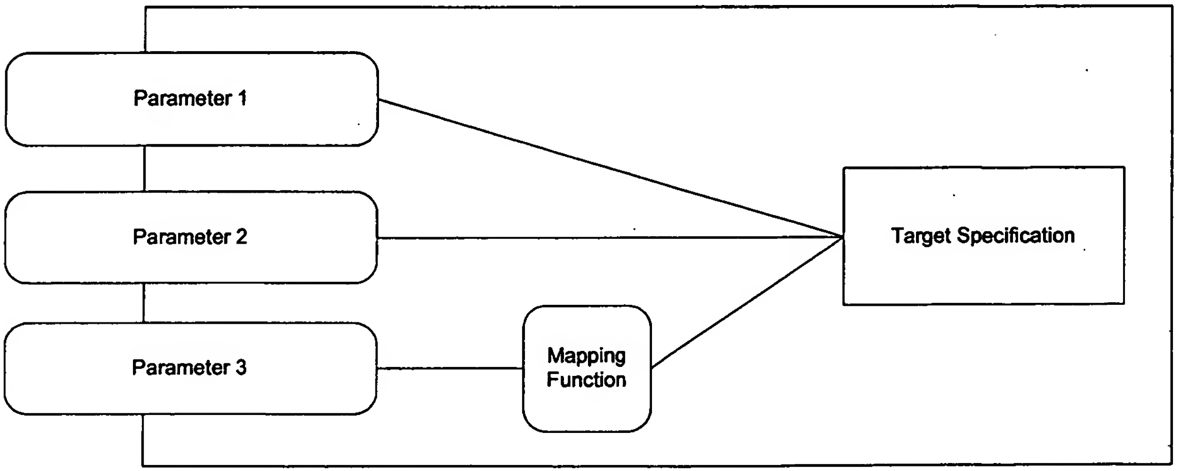
Simply put the URL for drill-through will be as closely implemented as with IWR as possible. Advantage of course being that existing Series 7 products will have as minimal changes as possible to permit drill-through to CR1.

Drill-through Path out of a CR1 report

Another requirement for Baltic is the ability to be able to drill from a CR1 report to either another CR1 report or to an external application (URL). This will be accomplished with a similar scheme as drilling into CR1 but in reverse. As in the drill-in design, report columns will be mapped to a set of output parameters. These parameters will be used as input for the drill-target. In the simplest case (and V1 deliverable) it is a URL template. The URL template would be a tokenized format string along the lines of a printf() kind of format string. For example something like…

http://whatever/what.cgi?predicate=(%P1=%s, %P2=%s …)
Here P1 and P2 are placed with the parameter name and %s with there values.
What the URL template really would look like and all the rules as to how parameter substitution would be done is **TBD**.

Lets take for example the simplest case, drilling from one CR1 report to another. No parameters are explicitly defined so all parameter-to-column mapping is done via the query element or column names (implicitly). In this case the drill target is another CR1 report. The output parameters required could be derived from the input parameters defined either explicitly or implicitly from the target report. Input parameters from the target report are matched up with potential output parameters from the source report. A URL is formed and sent off. The same rules that define what are eligible input parameters are likely going to be the same as the rules for eligible output parameter.



The Target application type determines the target specification. In the case of a CR1 report then the Target Specification is nothing more than a URL template where the parameter values can be plugged into. In the case of target application like PowerPlay web it is also a URL target with placeholder that indicate where the parameters/value pairs should go.

BMT's Role in Drill-through

BMT will extend the current CR1 model to include new meta-data to defined the following…

- Mapping parameters to model query items
- Defining Target Specifications (URL templates)

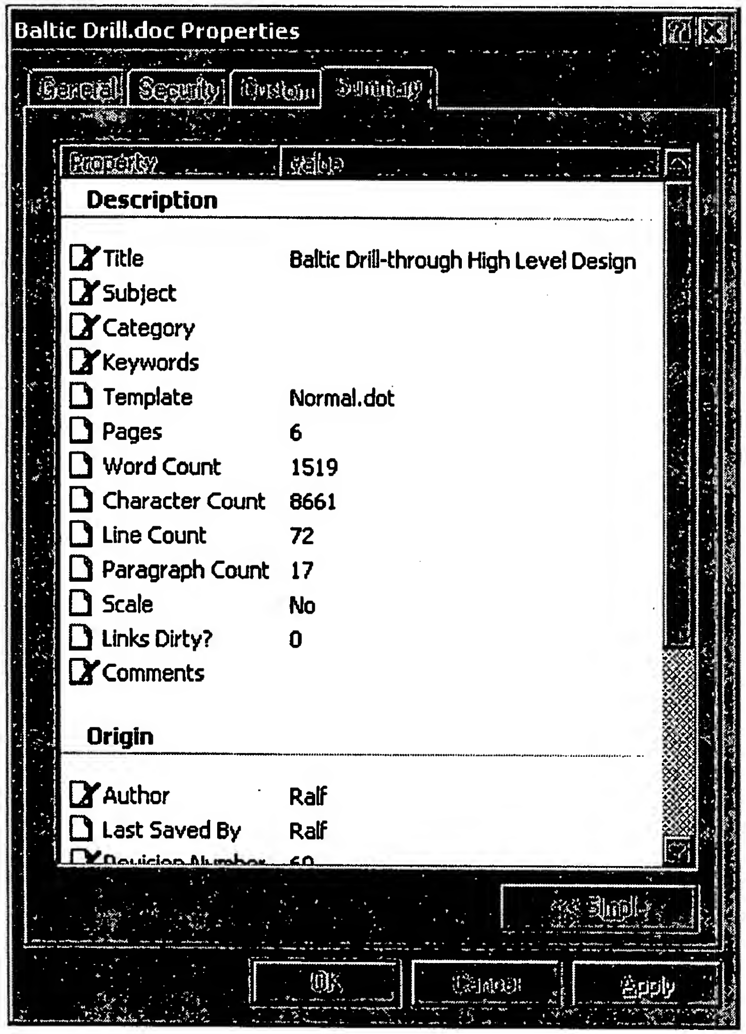
Publishing an IQD file

TBD

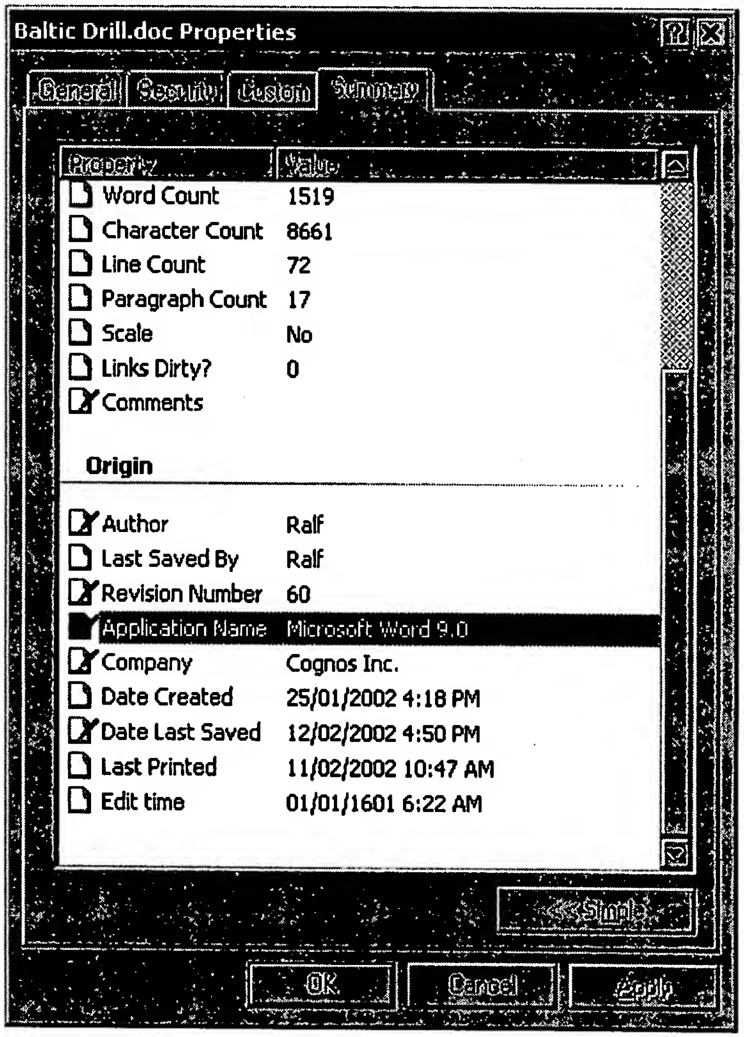
It might be advantageous if BMT could create IQD files from query specifications. This could be implemented as a type deployment. The advantage would be that cubes built from these IQD files via Transformer would have the column names that would match the meta-data that the target reports may be based on.

Along these same lines it may also be useful to me able to create IQD files from Query Studio based on a selected report.

It still needs to be determined by product management if this functionality is required.



Property	Value
Description	
☒ Title	Baltic Drill-through High Level Design
☒ Subject	
☒ Category	
☒ Keywords	
□ Template	Normal.dot
□ Pages	6
□ Word Count	1519
□ Character Count	8661
□ Line Count	72
□ Paragraph Count	17
□ Scale	No
□ Links Dirty?	0
☒ Comments	
Origin	
☒ Author	Ralf
□ Last Saved By	Ralf
☒ Revision Number	60



Property	Value
□ Word Count	1519
□ Character Count	8661
□ Line Count	72
□ Paragraph Count	17
□ Scale	No
□ Links Dirty?	0
☒ Comments	
Origin	
☒ Author	Ralf
□ Last Saved By	Ralf
☒ Revision Number	60
☒ Application Name	Microsoft Word 9.0
☒ Company	Cognos Inc.
□ Date Created	25/01/2002 4:18 PM
☒ Date Last Saved	12/02/2002 4:50 PM
□ Last Printed	11/02/2002 10:47 AM
□ Edit time	01/01/1601 6:22 AM

Exhibit D

Parameterized drill-through using data expressions within a static drill-through Model.

Design originators:
Ralf Vierich
Kevin Ferguson

Drill-through Concept

Drill-through is defined as the action of navigating from one report or dataview to another report or dataview and applying the context of the source to the target.
For example invoking a drill-through operation from a row of the source report to a target report, a filter is constructed based on the source report row and applied to the target report.

Problems we are trying to solve:

Previous releases, Cognos products relied on some very strict rules in order to effectively drill from one Cognos application to another. The drill-through filter generated from the drill-through source needed to match up with the column names used in the target report. This name matching was done through a set of intermediate files called IQD's. An IQD (Impromptu Query Definition) file is essentially the sql query that can be used by Transformer to generate the powercube. The IQD file contained the column name mapping that was used to generate the context filter that the target application could understand. The process is essentially the same with IWR to CognosQuery but the intermediate files where slightly different but still served as a column map.

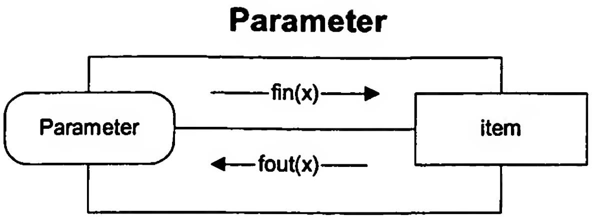
Solution in new a Drill-through Architecture

Although a drill-through path was implied in the previous product releases it was never exposed to the same level as in the new architecture.

A drill-through path is defined as collection of parameters that map context elements from the drill-through source (report) to the inputs of the target.

There are several concepts that are used in the new drill-through scheme.

The first is a parameter. A parameter is a way of exposing and naming a columns or more generally an input or output to a drill-through source or target. (A drill-through source or target could be a report or another application, URL what ever.)
A parameter exposes a drill-through "item" to the outside world and also allows the ability to add conversion functions that may act as data converters or filters.

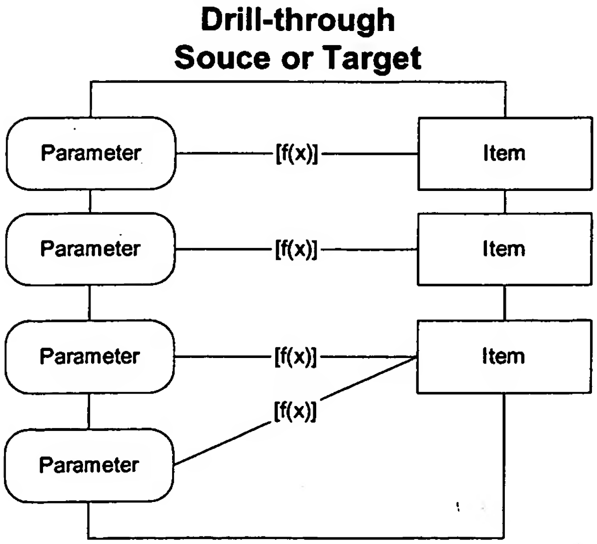


In the above diagram we show that an item which is the represents a report column or parameter or what ever from a drill-through source or target has a parameter exposed. The mapping from the item also has optionally two functions which may translate the date from the outside world (through the parameter) to the item and vise-versa. Mapping functions are optional.

Mapping functions

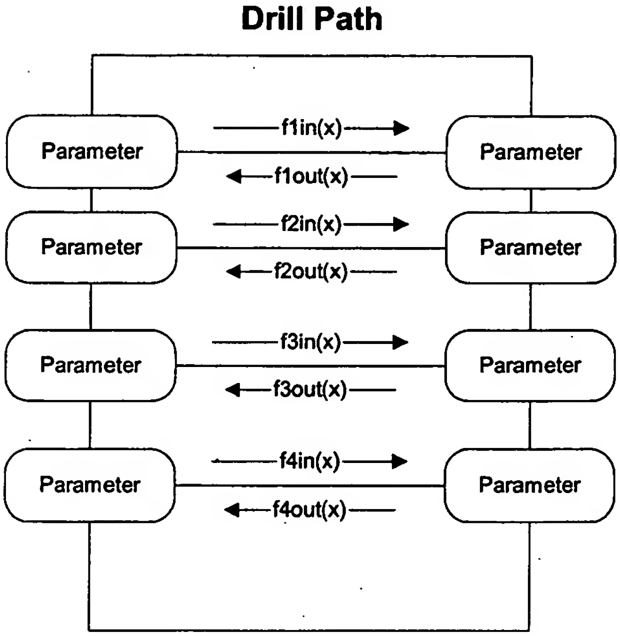
In previous release data conversion or manipulation of drill-through parameters could only be done within the report. This meant that the report would be likely only used specifically for drill-through since the report column used in the context filter would only make sense for drill-through.
By moving the input and output conversion functions into the drill-through model can leave the report alone and encapsulate any drill-through specific data conversions to a drill-through service.

More than one parameter can map to a single drill-though item.



So a drill-through source or target is generalized to simply a collection of parameters as show in the diagram above.

The next concept is a drill-through path. A drill-through path is a collection of source and target parameters. The drill-through path basically joins the source and target objects together. A single drill-though target can have many drill-through sources and vise-versa.



All of this is captured and modeled from a single administration application. A single drill-through service is provided that may be queried for drill-through information such as a list of targets from a given source or drill-through actions to be performed.

Why is it so special or what makes this so unique

This system solved several problems. First off, since the drill-through source and targets are generalized as a collection of input and output parameters, what the drill-through target is, is not important. For example it means that a PowerCubes build from the data extracted from database "A" can still drill-through to a report who's data comes from database "B". It is the modeler's responsibility to ensure that the drill-through paths from the source and targets are properly matched up and the appropriate conversion functions are added.

Generalize drill-through target specification

Along with this new drill-through design we also expose the functionality to allow the drill-through model designed to define a URL or FORM template that will allow products to drill-through to any web based Target.

For example lets take the case where a template is a string that contains one or more predefined drill-through parameter placeholders.

If the placeholder is defined as <P1>, <P2>... etc. we could define a drill-through target as a URL like...

http://myserver/path/app.cgi?params=(<P1>,<P2>,<P3>)

Lets say we have a report that contains 3 columns named C1, C2, and C3. A drill-through path is defined that maps this source with the URL target. In this case the drill-through path is (C1->P1, C2->P2, C3->P3). Drilling from the source report at a specific row to the target means that the row values for C1, C2 and C3 will be substituted into the URL at the P1, P2, and P3 placeholders.

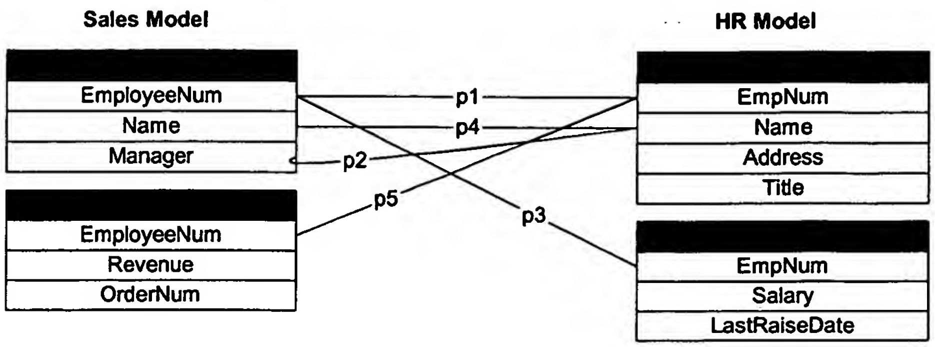
Drill-through paths between meta-data models

This section describes the concept of defining one or more drill-through paths between meta-data models. A meta-data model is the model in which the reports are based on. For example BMT is a meta-data modeling tool which is used by Report Net to build reports from.

In the simplest case, with the new drill-through model, a drill-through model designer can define a drill-through path from one report to another. This defines the drill-through parameters that will be used in the context filter when navigation from one report to another.
The new drill-through scheme also supports the concept of defining one or more drill-through paths between the different meta-data models. The advantage of supporting this is that reports based on the meta-data models can make implicit use of the drill-through paths. (This is a generalized way of assigning drill-through paths between reports without having to explicitly define drill-through paths between all reports.)

What we do is provide a list of potential drill-through paths between model query items.

Lets take the following use-case.
There are two different meta-data models. One model's the "Sales" database and is used to allow report writer to generate reports on all sales related information. Another model is based on the "Human Resources" database and allows report writers to generate reports of Personal information.
In this use-case it may be useful to allow managers to drill-through from sales reports to personal reports to answer questions like; "Am I paying this guy enough based on his sales record ?"



The modeler has connected the parameters between the models as follows (in the diagram above. P1 to P5 represent the parameter mapping from one model to the other. Note that some of the query items do not have parameters assigned to then (like Title, LastRaiseDate). For simplicity sake let us say that parameter mapping functions are not required.

Lets say now we have a source report based on the Sales model with the following columns;
[SalesEmployee].[Manager], [SalesEmployee].[Name], total([SalesMetric].[Revenue])

The target report is based on the HR Model with the following report columns;
[PersonalInfo].[Name], [PersonalInfo].[Title], [PersonalInfo].[Salary], [PersonalInfo].[LastRaiseDate]

The report authoring tool (in this case ReportNet) will ask the drill-through service for a list of pontential drill-through paths that could be used to drill-through from the source report (Sales) to the target report (HR Info).

The drill-through service employs the following algorithm to determine the paths;

1. gather list of parameters (query items) from the source and target reports.
2. from each source parameter look for a parameter path that map the parameter to the target. All parameter maps are collected as a single drill-through path
3. If more than one parameter path points to the same target parameter then duplicate the drill-path one for each duplicate target path.
4. Continue splitting until all the parameter maps for each drill-through path point to unique target parameters.

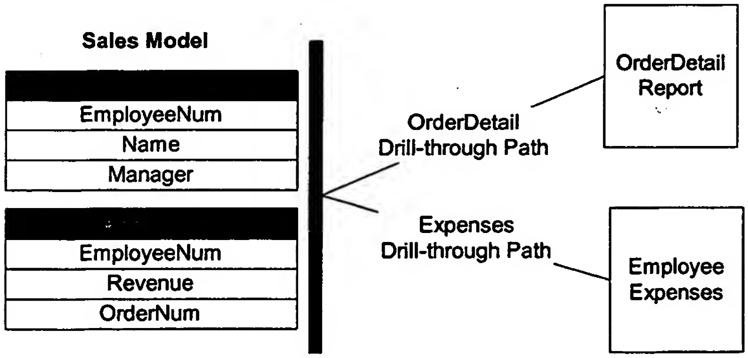
Following this algorithm the following drill-through paths will be returned from the drill-through service. Drill-though #1 (p4) and Drill-through #2 (p2). Therefore using this example we are saying that from the sales report you may drill-through to the HR report and filter on the salespersons name or you may drill-through filtering on the Managers name. Therefore potentially answering two questions; how much am I paying a sales person based on their generated revenue? Or how much am I paying a manager based the total revenue the sales people being managed?

(Note: in this example only a single parameter mapping was used since only one parameter was matched from source to target. A drill path may contain any number of applicable parameter mappings. The drill-through path splitting would occur as often as required.)

Model to Report Drill-through

Expanding on the general model drill-through usage. It is also possible to define general drill-through paths from a meta-data model to specific reports. As seen the diagram below the Sales model has 2 drill-though paths defined to 2 separate reports.
The "OrderDetail" drill-through path contains a single parameter mapping on [SalesMetrics].[OrderNum] to report column "Order Number". The "Expenses" drill-through path contains a single parameter mapping on [SalesEmployee].[EmployeeNum] to report column "Employee Number"

This means that any report based on the Sales model which includes report columns based on [SalesMetrics].[OrderNum] can drill-through to the OrderDetail Report. Similarly any report that includes a report column based on [SalesEmployee].[EmployeeNum] can drill-through to the Employee Expenses report.



Cognos OLAP to Model Drill-through

This section will describe the drill-through mapping to and from OLAP sources (all source via Powerplay and modeled with Transformer) to models for relational reporting tools (ReportNet)
Basically the mechinism is the same as described above. Levels and categories are exposes as parameters and can be mapped to model query items (parameters). They can also be mapped to reports.
{ more to come here I guess }

Parameterized drill-through using data expressions within a static drill-through Model.

Design originators:
Ralf Vierich
Kevin Ferguson

Drill-through Concept

Drill-through is defined as the action of navigating from one report or dataview to another report or dataview and applying the context of the source to the target.
For example invoking a drill-through operation from a row of the source report to a target report, a filter is constructed based on the source report row and applied to the target report.

Problems we are trying to solve:

Previous releases, Cognos products relied on some very strict rules in order to effectively drill from one Cognos application to another. The drill-through filter generated from the drill-through source needed to match up with the column names used in the target report. This name matching was done through a set of intermediate files called IQD's. An IQD (Impromptu Query Definition) file is essentially the sql query that can be used by Transformer to generate the powercube. The IQD file contained the column name mapping that was used to generate the context filter that the target application could understand. The process is essentially the same with IWR to CognosQuery but the intermediate files where slightly different but still served as a column map.

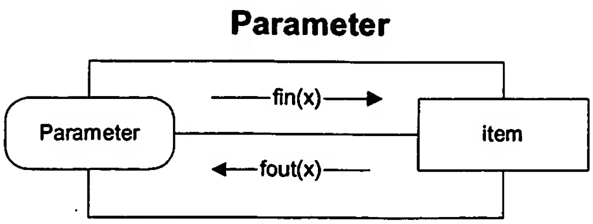
Solution in new a Drill-through Architecture

Although a drill-through path was implied in the previous product releases it was never exposed to the same level as in the new architecture.

A drill-through path is defined as collection of parameters that map context elements from the drill-through source (report) to the inputs of the target.

There are several concepts that are used in the new drill-through scheme.

The first is a parameter. A parameter is a way of exposing and naming a columns or more generally an input or output to a drill-through source or target. (A drill-through source or target could be a report or another application, URL what ever.)
A parameter exposes a drill-through "item" to the outside world and also allows the ability to add conversion functions that may act as data converters or filters.



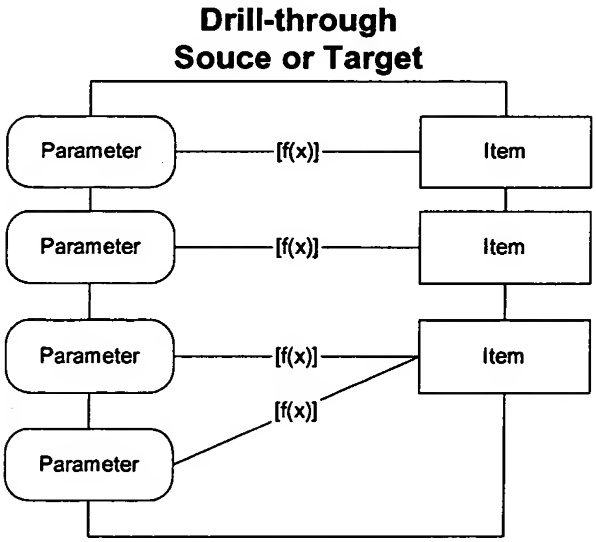
In the above diagram we show that an item which is the represents a report column or parameter or what ever from a drill-through source or target has a parameter exposed. The mapping from the item also has optionally two functions which may translate the date from the outside world (through the parameter) to the item and vise-versa. Mapping functions are optional.

Mapping functions

In previous release data conversion or manipulation of drill-through parameters could only be done within the report. This meant that the report would be likely only used specifically for drill-through since the report column used in the context filter would only make sense for drill-through.

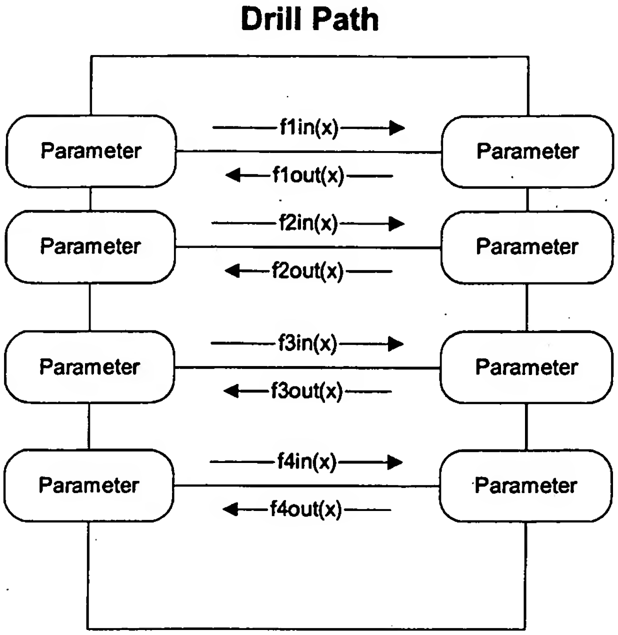
By moving the input and output conversion functions into the drill-through model can leave the report alone and encapsulate any drill-through specific data conversions to a drill-through service.

More than one parameter can map to a single drill-though item.



So a drill-through source or target is generalized to simply a collection of parameters as show in the diagram above.

The next concept is a drill-through path. A drill-through path is a collection of source and target parameters. The drill-through path basically joins the source and target objects together. A single drill-though target can have many drill-through sources and vise-versa.



All of this is captured and modeled from a single administration application. A single drill-through service is provided that may be queried for drill-through information such as a list of targets from a given source or drill-through actions to be performed.

Why is it so special or what makes this so unique

This system solved several problems. First off, since the drill-through source and targets are generalized as a collection of input and output parameters, what the drill-through target is, is not important. For example it means that a PowerCubes build from the data extracted from database "A" can still drill-through to a report who's data comes from database "B". It is the modeler's responsibility to ensure that the drill-through paths from the source and targets are properly matched up and the appropriate conversion functions are added.

Generalize drill-through target specification

Along with this new drill-through design we also expose the functionality to allow the drill-through model designed to define a URL or FORM template that will allow products to drill-through to any web based Target.

For example lets take the case where a template is a string that contains one or more predefined drill-through parameter placeholders.

If the placeholder is defined as <P1>, <P2>... etc. we could define a drill-through target as a URL like...

http://myserver/path/app.cgi?params=(<P1>,<P2>,<P3>)

Lets say we have a report that contains 3 columns named C1, C2, and C3. A drill-through path is defined that maps this source with the URL target. In this case the drill-through path is (C1->P1, C2->P2, C3->P3). Drilling from the source report at a specific row to the target means that the row values for C1, C2 and C3 will be substituted into the URL at the P1, P2, and P3 placeholders.

Drill-through paths between meta-data models

This section describes the concept of defining one or more drill-through paths between meta-data models. A meta-data model is the model in which the reports are based on. For example BMT is a meta-data modeling tool which is used by Report Net to build reports from.

In the simplest case, with the new drill-through model, a drill-through model designer can define a drill-through path from one report to another. This defines the drill-through parameters that will be used in the context filter when navigation from one report to another.

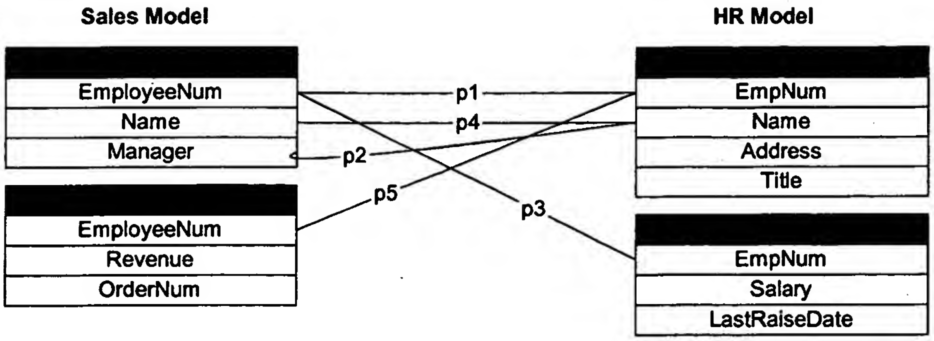
The new drill-through scheme also supports the concept of defining one or more drill-through paths between the different meta-data models. The advantage of supporting this is that reports based on the meta-data models can make implicit use of the drill-through paths. (This is a generalized way of assigning drill-through paths between reports without having to explicitly define drill-through paths between all reports.)

What we do is provide a list of potential drill-through paths between model query items.

Lets take the following use-case.

There are two different meta-data models. One model's the "Sales" database and is used to allow report writer to generate reports on all sales related information. Another model is based on the "Human Resources" database and allows report writers to generate reports of Personal information.

In this use-case it may be useful to allow managers to drill-through from sales reports to personal reports to answer questions like; "Am I paying this guy enough based on his sales record ?"



The modeler has connected the parameters between the models as follows (in the diagram above. P1 to P5 represent the parameter mapping from one model to the other. Note that some of the query items do not have parameters assigned to then (like Title, LastRaiseDate). For simplicity sake let us say that parameter mapping functions are not required.

Lets say now we have a source report based on the Sales model with the following columns;
[SalesEmployee].[Manager], [SalesEmployee].[Name], total([SalesMetric].[Revenue])

The target report is based on the HR Model with the following report columns;
[PersonalInfo].[Name], [PersonalInfo].[Title], [PersonalInfo].[Salary], [PersonalInfo].[LastRaiseDate]

The report authoring tool (in this case ReportNet) will ask the drill-through service for a list of pontential drill-through paths that could be used to drill-through from the source report (Sales) to the target report (HR Info).

The drill-through service employs the following algorithm to determine the paths;

1. gather list of parameters (query items) from the source and target reports.
2. from each source parameter look for a parameter path that map the parameter to the target. All parameter maps are collected as a single drill-through path
3. If more than one parameter path points to the same target parameter then duplicate the drill-path one for each duplicate target path.
4. Continue splitting until all the parameter maps for each drill-through path point to unique target parameters.

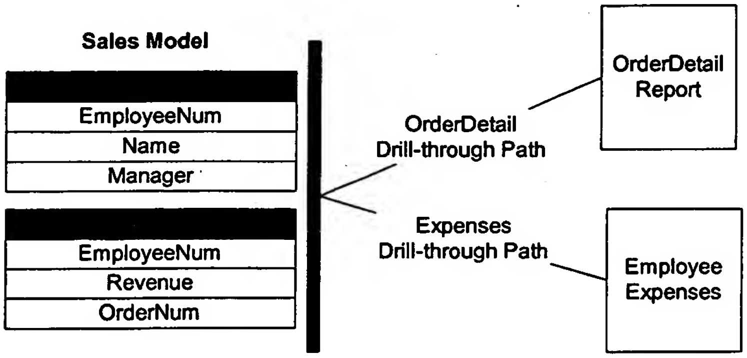
Following this algorithm the following drill-through paths will be returned from the drill-through service. Drill-though #1 (p4) and Drill-through #2 (p2). Therefore using this example we are saying that from the sales report you may drill-through to the HR report and filter on the salespersons name or you may drill-through filtering on the Managers name. Therefore potentially answering two questions; how much am I paying a sales person based on their generated revenue? Or how much am I paying a manager based the total revenue the sales people being managed?

(Note: in this example only a single parameter mapping was used since only one parameter was matched from source to target. A drill path may contain any number of applicable parameter mappings. The drill-through path splitting would occur as often as required.)

Model to Report Drill-through

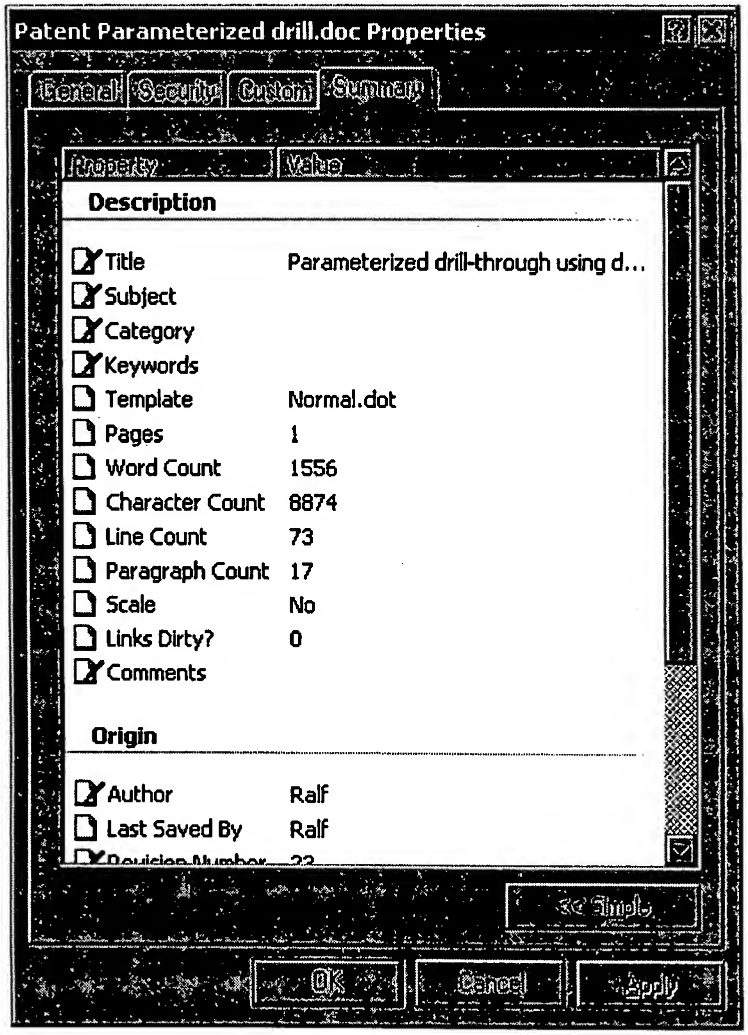
Expanding on the general model drill-through usage. It is also possible to define general drill-through paths from a meta-data model to specific reports. As seen the diagram below the Sales model has 2 drill-though paths defined to 2 separate reports.
The "OrderDetail" drill-through path contains a single parameter mapping on [SalesMetrics].[OrderNum] to report column "Order Number". The "Expenses" drill-through path contains a single parameter mapping on [SalesEmployee].[EmployeeNum] to report column "Employee Number"

This means that any report based on the Sales model which includes report columns based on [SalesMetrics].[OrderNum] can drill-through to the OrderDetail Report. Similarly any report that includes a report column based on [SalesEmployee].[EmployeeNum] can drill-through to the Employee Expenses report.



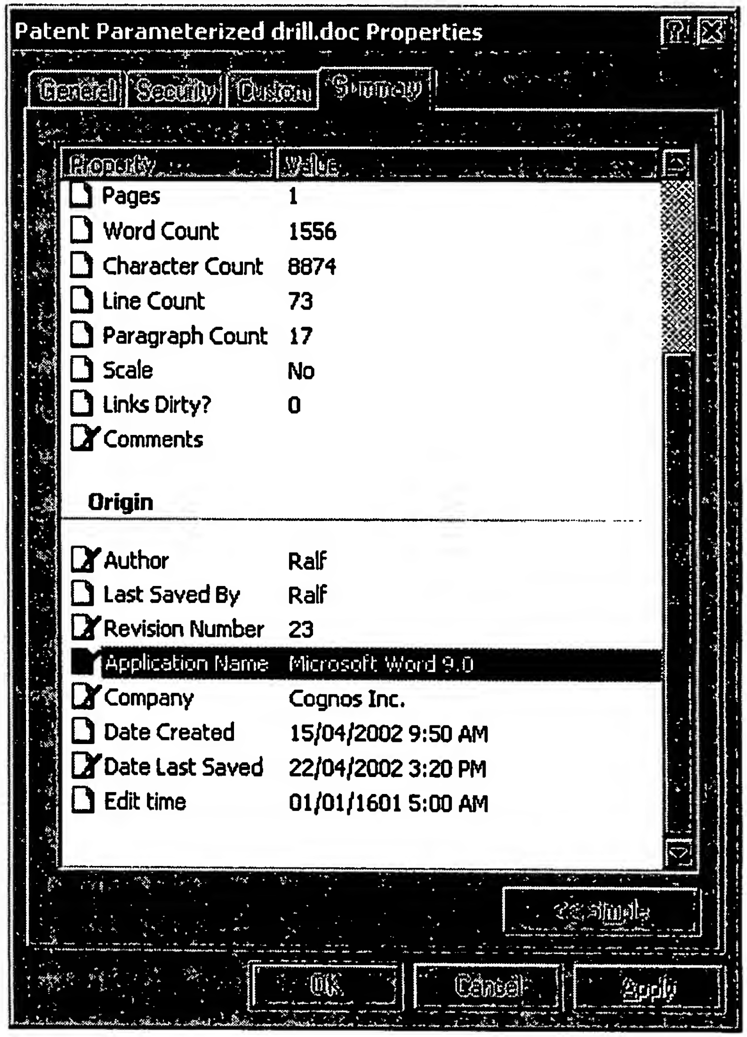
Cognos OLAP to Model Drill-through

This section will describe the drill-through mapping to and from OLAP sources (all source via Powerplay and modeled with Transformer) to models for relational reporting tools (ReportNet)
Basically the mechinism is the same as described above. Levels and categories are exposes as parameters and can be mapped to model query items (parameters). They can also be mapped to reports.
{ more to come here I guess }



Patent Parameterized drill.doc Properties

Property	Value
Description	
☑ Title	Parameterized drill-through using d…
☑ Subject	
☑ Category	
☑ Keywords	
□ Template	Normal.dot
□ Pages	1
□ Word Count	1556
□ Character Count	8874
□ Line Count	73
□ Paragraph Count	17
□ Scale	No
□ Links Dirty?	0
☑ Comments	
Origin	
☑ Author	Ralf
□ Last Saved By	Ralf
☑ Revision Number	23



Patent Parameterized drill.doc Properties

Property	Value
□ Pages	1
□ Word Count	1556
□ Character Count	8874
□ Line Count	73
□ Paragraph Count	17
□ Scale	No
□ Links Dirty?	0
☑ Comments	
Origin	
☑ Author	Ralf
□ Last Saved By	Ralf
☑ Revision Number	23
☑ Application Name	Microsoft Word 9.0
☑ Company	Cognos Inc.
□ Date Created	15/04/2002 9:50 AM
☑ Date Last Saved	22/04/2002 3:20 PM
□ Edit time	01/01/1601 5:00 AM

Exhibit F